VŠB – Technická univerzita Ostrava
Fakulta Elektrotechniky a informatiky
Katedra Informatiky

Systém pro ovládání dveřního zámku za pomocí RFID karet
System for Remote Control of Door Lock with RFID Cards

2010 Tomáš Mollnhuber

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně, pouze za odborného vedení vedoucího bakalářské práce Ing. Davida Seidla.

Dále prohlašuji, že veškeré podklady, ze kterých jsem čerpal, jsou uvedeny v seznamu literatury.

V Ostravě dne 3. května 2010 .....

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Davidu Seidlovi za konzultace a cenné rady při návrhu softwaru, dále také za rady při psaní bakalářské práce.

Tomáš Mollnhuber

ABSTRAKT

V bakalářské práci se zabývám návrhem systému pro ovládání zámku za pomocí RFID karet. Práce stručně seznamuje s technologiemi, které jsou použity v návrhu sytému. Jedná se o protokol LDAP, bezdrátovou identifikaci RFID a převodník RS232 na Ethernet. Podrobněji je popsáno řešení jednotlivých části softwaru, např. komunikace s převodníkem, reprezentace dat z RFID čtečky, navázání spojení s LDAP serverem nebo programová implementace SMTP protokolu.

Klíčová slova: LDAP, RFID, XPort

ABSTRACT

In my bachelor thesis I deal with the project of the system used for controlling locks by RFID cards. The thesis gives information about the technologies used in the system. It deals with the protocol LDAP, wireless identification RFID and convertor from RS232 into the Ethernet. Different parts of the software are described in details there. For example the communication with the convertor, interpretation of information from the RFID scanner, connecting with the LDAP server or programm implementation of the SMTP protocol.

Key words: LDAP, RFID, XPort

Seznam zkratek

DAP	- Directory Access Protocol, protokol k přístupu k adresářovým službám
DIT	- Directory Information Tree, informace uloženy ve stromové struktuře
EPC	- Electronic Product Code, elektronické označení výrobku
LDAP	- Lightweight Directory Access Protocol, odlehčená verze DAP
PKI	- Public Key Infrastructure, infrastruktura veřejného klíče
RDN	- Relative Distinguished Name, relativní rozlišovací jméno
RFID	- Radio Frequency Identification, identifikace na radiové frekvenci
RS232	- sériový způsob komunikace
SASL	- Simple Authentication and Security Layer
SMTP	- Simple Mail Transfer Protocol, protokol pro posílaní e-mailů

Seznam ilustrací

Seznam tabulek

Obsah

1 Úvod

V dnešní moderní době počítačů a různých technologií v oblasti autentizace je otázka zda používání klasických klíčů není zastaralé. Zvláště ve velkých firmách, školách a univerzitách, kde zaměstnanci popřípadě studenti používají identifikační karty na různé služby, jako například hlídaní pracovní doby, objednáváni stravy atd., je nasnadě využít tyto identifikační karty k odemykání místnosti.

Z tohoto řešení plyne několik výhod - není třeba nosit s sebou svazek klíčů, stačí nám pouze identifikační karta, kterou otevřeme všechny místnosti do kterých máme přístup, místnosti jsou samozřejmě vybaveny čtecím zařízením. Další výhodou je možnost vytváření statistik o přístupu do určité místnosti. Dále si můžeme vytvářet jednoduché seznamy osob, které mají do dané místnosti povolen vstup a které nikoliv.

Cílem této bakalářské práce je vytvoření softwaru pro odemykaní učeben a místností pomocí čtecího zařízení a převodníku RS232 a Ethernet. Hardware, se kterým můj software spolupracuje, byl dodán panem Ing. Davidem Seidlem. Celý software je vytvořen v programovacím jazyce C++, pod operačním systémem Linux. Jazyk C++ jsem si vybral pro mojí znalost tohoto jazyka a velkého množství knihoven, které jsou k dispozici.

Celá bakalářská práce je rozdělena do 6 kapitol. V první kapitole je stručně popsána problematika adresářového serveru. V kapitole 3 je nastínění teorie čtecího zařízení a stručný popis použitého obvodu. Vlastnosti převodníku a jeho možnosti jsou popsané v kapitole 4. V této kapitole hovořím o nastavení převodníku tak, aby spolupracoval s vytvořeným softwarem. V předposlední kapitole 5 je popsaný celý systém jako celek. A nakonec kapitola 6 popisuje celý software.

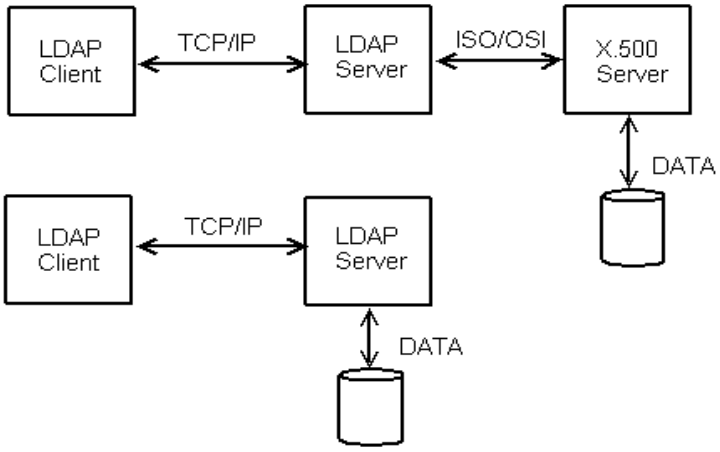
2 Adresářová služba LDAP

Adresářová služba (*directory service - DS*) je aplikace, která ukládá a organizuje informace o uživatelích a zdrojích v počítačových sítích. Tato služba přistupuje k adresáři, nebo může fungovat jako centrální autentizační autorita, která umožňuje autentizaci zdrojů.

Adresářová služba zprostředkovává informace z adresáře administrátorům, uživatelům, aplikacím atd. Dále také vytváří fyzickou síťovou topologii a protokoly. Adresářové služby jsou vhodné pro účely ve stylu kartotéky (správa uživatelů, telefonních čísel apod.). Naproti tomu mají některé nevýhody oproti relačním databázím například kontrola dat (referenční integrita), složité dotazy, složité modifikace. Rozšířeným příkladem adresářové služby je LDAP.

LDAP (Lightweight Directory Acces Protocol) byl původně navržen jako zjednodušená varianta protokolu DAP (directory access protocol), tj. jako jednodušší přístupový protokol mezi klientem a adresářovým serverem (X.500). Hlavní zjednodušení spočívá ve využití protokolů TCP/IP jako komunikační vrstvy (minimalizovány operace, vypuštění složitých vlastností).

Postupem času se protokol LDAP osamostatnil, byl vytvořen samostatný LDAP server viz. kresba 1, což znamená, že LDAP neslouží pouze jako přístupový protokol k X.500 serveru. Z hlediska klienta by mělo být jedno, zda přistupuje k adresářovým službám realizovaným samostatným LDAP serverem nebo LDAP server slouží jako brána k X.500 serveru. Příkladem implementace adresářových služeb LDAP je Active Directory od firmy Microsoft, nebo svobodná implementace OpenLDAP.



Kresba 1: Protokol LDAP

2.1 Architektura LDAP protokolu

Protokolem LDAP je definovaná komunikace mezi serverem a klientem. Klient pošle serveru zprávu, která obsahuje požadavek k provedení jedné z definovaných operací, načež server vrátí odpověď.

Základní schéma komunikace:

- Klient naváže spojeni s LDAP serverem. Pro operaci navázaní spojení se

používá termín *binding*. Klient má možnost se připojit anonymně, nebo se provede autentizace (klient prokáže svoji identitu).

- Pokud klient navázal spojení s LDAP serverem potom může v rámci spojení posílat žádosti o provedení operací nad adresářovými daty.
- V poslední fázi klient uzavře spojení s adresářovým serverem. Pro tuto operaci se používá termín *unbinding*.

LDAP je popsán pomocí následujících čtyř modelů:

- Informační model – struktura informací uložených v adresářových službách.
- Jmenný model – organizace a identifikace informací v adresářových službách.
- Funkční model – operace nad informacemi v adresářových službách.
- Bezpečnostní model – řízení přístupu k informacím v adresářových službách.

2.3 Informační model

Informace v adresáři jsou uloženy ve stromové struktuře, která se nazývá *Directory Information Tree* (DIT). Kořenem adresářového stromu je *rootDSE,* který obsahuje globální informace o adresáři. Podstata informačního modelu je založena na záznamech, které obsahují informace o různých objektech (uživatel, počítač atd.). LDAP záznamům se říká *objekt.* Tyto objekty se skládají ze skupiny *atributů*, které mají vždy typ a jednu nebo více hodnot.

Schéma určuje implementaci informačního modelu. Je to sada objektů, které definují strukturu a obsah každého objektu, který by mohl být vytvořen v adresářové službě. Můžeme říct, že schéma definuje všechny třídy objektů a atributy, které se mohou vyskytovat v adresářové službě.

Třídy objektů (object classes) jsou kategorie objektů, které můžou být vytvořeny v adresáři. V LDAP serveru se používá označení *objectClasses* a může se například jednat o *user, computer*, *domain*, *group* atd.. Třídy objektů jsou zařazeny do jedné z následujících tří skupin *Structural, Abstract* a *Auxiliary*.

Jelikož může nastat situace, kdy objekt je zařazen do více objectClasses, mohlo by být vyhledávání méně efektivní a ne zcela přesné. Z tohoto důvodu můžeme použít hledání podle kategorie objektů (*objectCategory*). *ObjectCategory* má, na rozdíl od objectClasses, pouze jednu hodnotu, což by mělo ukazovat na specifickou třídu v hierarchii tříd objektů.

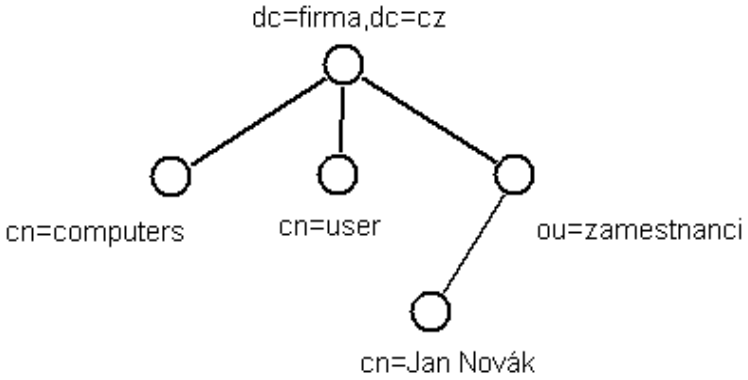
Atributy objektů (*object attibutes*) jsou vlastnosti objektů. Atribut může obsahovat jednu nebo více hodnot (jméno, příjmení, e-mail). Schéma určuje, které hodnoty atributů musí být vyplněné a které jsou nepovinné.

2.4 Jmenný model

Pro jednoznačnou identifikaci objektu a popis úplné cesty k záznamu (pozici ve stromě) se používá *Distinguished Name (DN). DN* obsahuje jména objektu a jména jednotlivých kontejnerů a domén, které obsahují objekty oddělené čárkou. Jednotlivé položky obsahují název atributu a přiřazenou hodnotu atributu. Na kresbě 2 je příklad

zaměstnance Jana Nováka, pro kterého je

„*DN = cn=Jan Novák,ou=zamestnanci,dc=firma,dc=cz*“.



Kresba 2: Jmenný model

Někdy nemusíme specifikovat celou cestu k objektu, potom můžeme použít *Relative Distinguished Name (RDN)*, která je relativní a jednoznačná v daném kontejneru. Jedná se o nejspodnější část DN, pro našeho zaměstnance je to *RDN = cn= Jan Novák*.

Kromě DN a RDN můžeme také použít OID (*Object Identifiers*), což znamená hierarchický, unikátní identifikátor, složený s dekadických číslic oddělených tečkou. Tento identifikátor je běžný u X.500.

Kterákoliv část DN je udávána jako *typ_atributu=hodnota.* Typ atributu, který se používá k popisu RD, se nazývá *jmenný atribut.* Každá třída má přiřazen jmenný atribut, jak je znázorněno v tabulce 1.

LDAP atribut	***Jméno***
CN	Common Name
OU	Organization Unit
O	Organization
C	Country

Tabulka 1: Třídy atributů

2.5 Funkční model

Funkční model LDAP serveru přesně popisuje, jaké operace můžeme provádět s informacemi v adresáři. Jedná se o 9 operací, které jsou rozděleny do 3 funkčních oblastí viz. tabulka 2.

Oblast	*Operace*	*Popis*
autentizace (Authentication)	bind	inicializuje spojení, vyjednává o metodě autentizace, autentizuje
	unbind	ukončí session
	abandon	klient žádá o ukončení posílání výsledků na poslední dotaz
dotazování (Interrogation)	search	výběr dat z určitého regionu pomocí filtru
	compare	porovná hodnotu atributu se zadanou hodnotou
aktualizace (Update)	add	vytvoří nový objekt
	modify	upraví atributy záznamu (vytvořit, smazat, upravit)
	modify RDN	slouží k přesunutí objektu v rámci stromu adresáře
	delete	smazání záznamu

Tabulka 2: Popis operací LDAP serveru

2.6 Bezpečnostní model

Úkolem bezpečnostního modelu je zamezení přístupu neoprávněné osoby k informacím v adresáři. Uživatel může být autentizován několika způsoby.

- Anonymní autentizace – v operaci *bind* nejsou žádné informace o identitě nebo heslu uživatele.
- Jednoduchá autentizace – pomocí DN a hesla. Tuto autentizaci je možné provádět po bezpečném kanále TLS/SSL. V této variantě nejprve dojte k výměně certifikátu obou stran a po ověření těchto certifikátů dojde k otevření spojení.
- Proxy autentizace – využívá existence definovaného uživatele, který má oprávnění nahlížet na hesla ostatních uživatelů. Autentizace uživatele probíhá přes tento mezičlánek.
- PKI autentizace – pracuje na principu PKI (*Public Key Infrastructure)* digitálních certifikátu, které jsou uloženy v atributu *userCertificate.* Pokud uživatel žádá o přístup k LDAP serveru, musí zadat své heslo, poté je autentizován po srovnání digitálních certifikátů na straně klienta a serveru.
- SASL (*Simple Authentication and Security Layer*) autentizace – jedná se o rozhraní, jehož prostřednictvím lze propojovat standardní mechanizmy s komunikačními protokoly. Podpora SASL je novou vlastností standartu LDAP verze 3.

Autorizace blízce souvisí s mechanizmem autentizace a nastupuje po jejím úspěšném ukončení. Úkolem autorizace je zajistit, aby měl autentizovaný uživatel přístup pouze k datům a operacím, ke kterým má oprávnění. Tato problematika se týká přístupových práv k záznamům a atributům. Přístupová práva můžeme nastavit až na úroveň jednotlivých atributů.

3 Bezdrátová identifikace RFID

RFID (*Radio Frequency Identification*) nebo také identifikace na radiové frekvenci je generace identifikačních čipů. Tyto čipy jsou v provedení pro čtení nebo pro zápis. Pracují převážně na nosné frekvenci 125 kHz, 134 kHz a 13,56kHz.

3.1 Typy RFID čipů

Pasivní

RFID snímač periodicky vysílá pulzy do okolí. Jestliže se v blízkosti objeví pasivní RFID čip, využije přijímaný signál k nabití svého napájecího kondenzátoru a odešle odpověď. Tyto čipy můžou buď vysílat jedno číslo EPC (Electronic product code), nebo obsahují navíc dodatečnou paměť, do které lze zapisovat další informace.

Aktivní

Používá se v menším měřítku než pasivní systém RFID, protože jsou složitější a také dražší, jelikož obsahují navíc napájecí zdroj a jsou schopny samy vysílat své identifikace. Z tohoto důvodu jsou používány pro aktivní lokalizaci. Dále aktivní RFID čtečky mají prostor pro další informace, které dokáží ukládat nebo odeslat spolu s identifikačním číslem.

3.2 Identifikace RFID čipů

RFID čipy obsahují 96bitové jedinečné číslo EPC, které může být přiděleno každému jednotlivému čipu. EPC se přiděluje centrálně výrobcům v jednotlivých řadách. EPC o délce 96 bitů může nabídnout číselný prostor 268 milionům výrobcům, produkujících 16 milionů druhů výrobku (tříd) a v každé třídě je prostor pro 68 miliard sériových čísel.

3.3 Hybridní obvod pro čtení pasivních RFID

RFID čtečka série AXA012 je hybridní obvod pro čtení bezkontaktních identifikátoru na frekvenci 125 kHz. Pří vložení bezkontaktního identifikátoru do elektromagnetického pole čtečky AXA012 je na výstup čtečky vyslán přímo identifikační kód bezkontaktního identifikátoru. Identifikační kód má tvar 16 bajtového pole dat, jak znázorňuje kresba 3.

STX (02H)	DATA (10 ASCII)	CHECKSUM (2 ASCII)	CR	LF	ETX (03H)

Kresba 3: Struktura dat z obvodu AXA012

- STX (Start of Text) – indikuje začátek textu
- DATA – 10 bajtů každý obsahuje jeden ASCII znak
- CHECKSUM – 2 bajty, je to vlastně XOR všech znaků dat.
- CR (*Carriage Return*) – speciální znak, který posune kurzor na začátek řádku.
- LF (*Line feed*) - řídicí znak, který posune kurzor na další řádek
- ETX (End of Text) - indikuje konec textu.

4 Převodník linek RS232 TTL na Ethernet

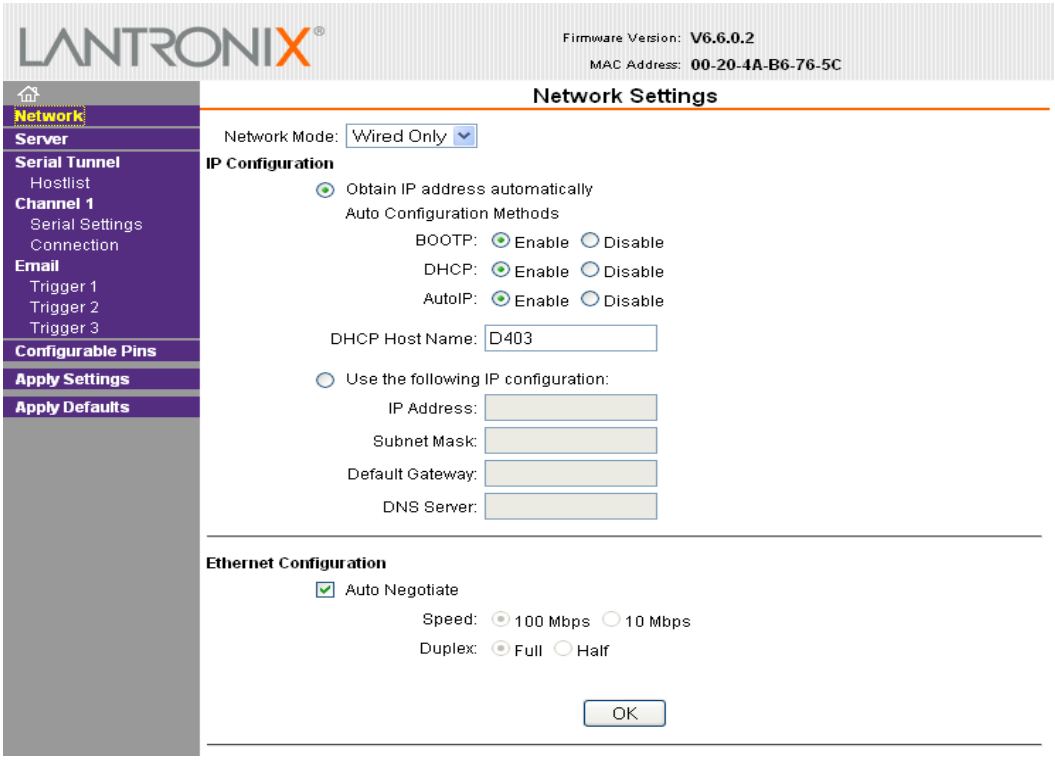
V zapojení je použit jako převodník obvod XPort. Tento obvod je kompletní převodník Ethernet – RS232 TTL (Sériová linka). Umožňuje snadné připojení zařízení k Ethernetu, pouze připojíme sériovou linku přes tento modul k Ethernetové sítí. Díky tomuto modulu je snadné připojit zařízení k Internetu nebo firemnímu Intranetu.

4.1 Vlastnosti XPortu

- Umožňuje protokoly: TCP/IP, UDP/IP, ARP, ICMP, SNMP, TFTP, Telnet, DHCP, BOOTP, HTTP, AutoIP.
- Obsahuje interní WEBové stránky (384 kB).
- Umožňuje odesílání e-mailu.
- K dispozici je varianta s šifrováním (256-bit AES Rijndael).
- Ethernet 10Base-T nebo 100Base-TX (Automatické rozpoznání).
- Obsahuje výkonný procesor (12 MIPS, 48MHz nebo 88MHz, architektura x86).
- Je napájen 3,3V.

4.2 Nastavení XPortu

Aby XPort pracoval tak jak je požadováno, musel jsem nastavit jeho základní konfiguraci. Jak je uvedeno výše, XPort obsahuje interní WEBové stránky, přes které lze nastavit vlastnosti XPortu. Na tyto stránky jsem se dostal po zadání IP adresy XPortu do webového prohlížeče. Zde v sekci *Network* jsem zaškrtl možnost *Obtain IP address automatically* a do textového pole *DHCP Host Name:* jsem vyplnil jméno XPortu jak je vidět na kresbě 4.

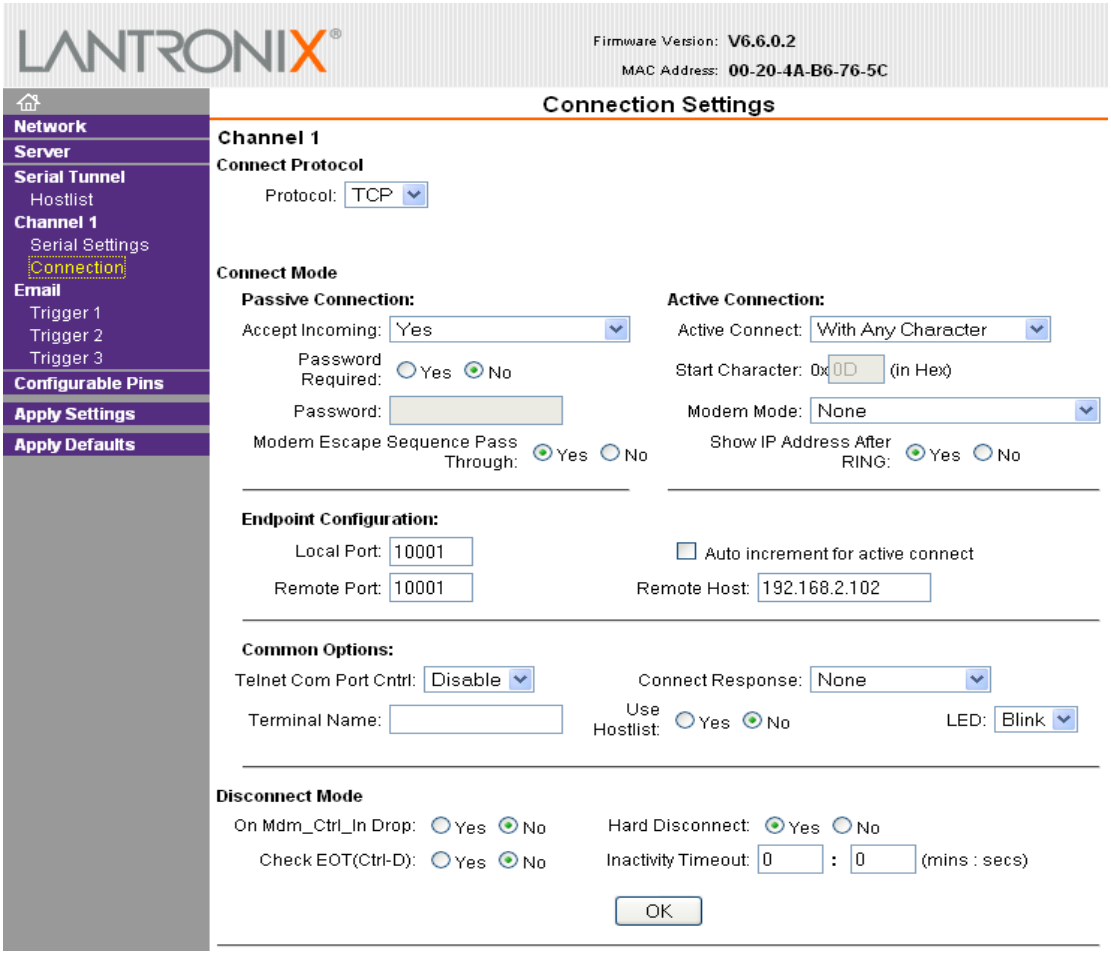


Kresba 4: XPort nastavení sítě

DHCP Host name (jméno XPortu) by mělo odpovídat učebně nebo místností, pro kterou bude XPort plnit funkci odemykání. Na jméno se bude také odkazovat v konfiguračním souboru programu, kde pro každý XPort bude zvlášť nastavení.

Dále jsem pokračoval v nastavování XPortu v sekci *Connection*, kde jsem nastavil *Protocol* na TCP, *Remote host* na adresu serveru, kde se bude XPort připojovat v případě aktivace z RFID čtečky. V poslední řadě jsme nastavil *Remote port* a *Local port,* jak je patrné z kresby 5. Remote port znamená číslo portu serveru, na který se bude připojovat XPort, a Local port je číslo XPortu, na který se bude připojovat server. Nutno dodat, že čísla portu se předělují většinou z rozsahu 1 023 - 65 535, menší čísla jsou přidělena nejběžnějším službám.

Toto nastavení plně vyhovuje pro komunikaci se serverem, u XPortu toho však můžeme nastavit mnohem více, ale to není tématem mé bakalářské práce.



Kresba 5: XPort nastavení serveru

5 Popis zapojení

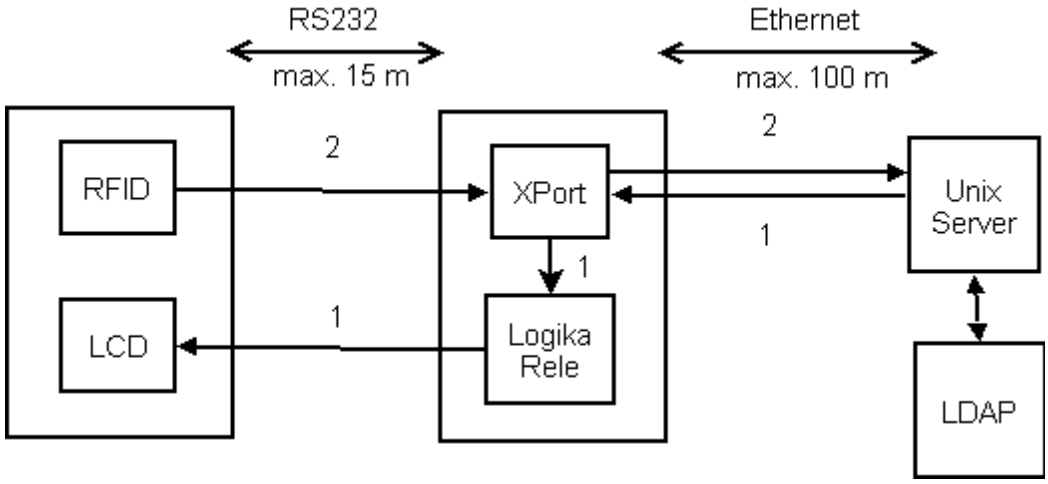
Na kresbě 6. je znázorněno blokové schéma celého zapojení včetně LDAP serveru. Celé zapojení se skládá z modulu RFID čtečky, modulu XPortu, programu Serveru a použitého LDAP serveru. Spojení mezi modulem RFID čtečky a XPortem musí mít podle standartu RS-232 maximální délku 15 metrů. Délka spoje mezi modulem XPortu a Ethernet zásuvkou je podle normy EEE 802.3 100 metrů.

Potenciální osoba vidí pouze RFID čtečku, která je u vstupních dveří učebny nebo místnosti, do kterých daná osoba hodlá vstoupit. Osoba přiloží svoji identifikační kartu k RFID čtečce, která sejme identifikační kód osoby a pošle tento kód XPortu. XPort převede data z RFID čtečky na Ethernet a vytvoří spojení na server. Po úspěšném vytvoření spojení zašle XPort data na server.

Server zkontroluje zda se data nepoškodila při přenosu a následně je dále zpracuje. Po zpracování, kdy se z dat vyjme identifikační řetězec, je pošle na LDAP server. Následně LDAP server z identifikačního řetězce zjistí informace o osobě, která se snaží o přístup do dané místnosti. Zpět serveru pošle LDAP server jméno a e-mail dané osoby.

Na základě těchto dat a konfiguračního souboru server vyhodnotí zda daná osoba má přístup do místnosti, do které chce vstoupit. Pokud osoba má oprávnění vstoupit do místnosti, pošle server příkaz XPortu na otevření dveří. Jestliže osoba nemá přístup do místnosti server nejenže nepošle příkaz na otevření dveří, ale také pošle e-mail o neoprávněném přístupu do místnosti osobě, která se pokoušela vstoupit do místnosti.

Dále server posílá zpět přes XPort data, která se průběžně zobrazují na displeji RFID čtečky. Jedná se o název místnosti nebo učebny, aktuální datum a čas a informace o výsledku požadavku vstupu do místnosti.



Kresba 6: Schéma zapojení

6 Návrh Softwaru

Návrh celého systému byl omezen jednou podmínkou a to, že bude vyvíjen pod operačním systémem Linux. Jako distribuci Linuxu jsem si vybral Ubuntu 8.10 a to jak pro její snadnou instalaci, tak pro silné spojení s komunitou Debian a z toho plynoucí používání formátu balíčku deb.

Dále bylo potřeba vybrat vhodný programovací jazyk pro návrh a vývoj celého systému. V podstatě jsem měl dvě možnosti - buď programovací jazyk JAVA nebo jazyk C++. Po důkladném zvážení jsem se rozhodl pro programovací jazyk C++. Tento programovací jazyk je rychlý, i když toto není rozhodující kritérium pro tento systém. Důležitým faktorem je velké množství knihoven pro vývoj mého systému.

Celý systém dohromady tvoří sedm tříd *Server.cpp*, *AutorizaceLDAP.cpp*, *Log.cpp*, *Conf.cpp*, *md5.cpp, Mail.cpp* a *md5wrapper.cpp*. Hlavní třída *Server.cpp,* jak už je patrné z názvu, je vlastně server, na který se připojují klienti v podobě XPortu. Tato třída má na starosti obousměrnou komunikaci s jednotlivými místnostmi, u kterých požadujeme autorizovaný vstup. Dále volá třídu *AutorizaceLDAP.cpp,* která má za úkol zjistit jestli osoba, která žádá o přístup do místnosti má povolení.

Tedy třída *AutorizaceLDAP.cpp* z řetězce, který jí předá třída *Server.cpp,* zjistí ze školního LDAP serveru informace o osobě, která se pokouší vstoupit do místnosti. Tyto informace porovná s konfiguračním souborem a výsledek předá zpět třídě *Server.cpp*. O inicializaci a konfiguraci konfiguračního souboru se stará třída *Conf.cpp*, která se stará také o porovnávaní, zda má osoba právo vstoupit do místnosti.

Pokud třída *Server.cpp* dostane zprávu, že daná osoba nemá přístup do dané místnosti zavolá třídu *Mail.cpp,* která informuje tuto osobu o neoprávněném přístupu prostřednictvím e-mailové zprávy. Dále je tu třída *Log.cpp,* která má za úkol zaznamenávat všechny události vzniklé během provozu programu a tyto události ukládat do speciálního souboru. Jedná se například o čase vstupu osob do místností nebo neoprávněném vstupu do místnosti.

Poslední dvě třídy *md5.cpp* a *md5wrapper.cpp* volá třída *Server.cpp* a slouží k hašovaní řetězce, který přijde z XPortu. Tento hašovaný řetězec je následně poslán na LDAP pomocí třídy *AutorizaceLDAP.cpp.* Jelikož jsou tyto dvě třídy volně dostupné na internetu, tak jsem se je nesnažil naprogramovat a použil jsem stávající.

V následujících kapitolách popisuji podrobněji jednotlivé řešení mého vytvořeného programu. Nebudu zde popisovat celý program, ale jen ty části, které jsou z mého pohledu zajímavé.

6.1 Mezi systémová komunikace pomocí soketů

Celý program je koncipován jako komunikace klient – server, kde program je server a každý XPort představuje klienta. Pro komunikaci mezi klientem a serverem jsme se rozhodl použít tzv. Sokety. Jedná se o obousměrnou meziprocesní a mezi systémovou komunikaci, kterou lze nastavit požadované parametry.

Soket se vytvoří v C++ následovně:

```
int socket_listen= socket(AF_INET, SOCK_STREAM, 0);
```

V mém případě má funkce socket() tři parametry. První parametr AF_INET znamená, že je použita rodina protokolu IP. Druhý parametr *SOCK_STREAM* odpovídá protokolu TCP, což znamená, že před vlastní komunikací se naváže spojení, čímž se obě strany ujistí, že jsou spolu ochotny komunikovat. Poslední parametr 0 znamená, že jsem nepoužil žádný protokol. Celá funkce vrátí file-deskriptor *socket_listen.*

Dále bylo nutné vytvořit strukturu typu sockaddr_in *server_addr*, která je společná pro všechny sokety a předává se jako parametr funkcím pracujícími se sokety.

```
hostent *hostip = gethostbyname(conf->getIP_Server("ip_server"));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(conf->getPort_Server("port"));
server_addr.sin_addr = *(in_addr *) hostip->h_addr_list[ 0 ];
```

Funkce *gethostbyname()* přeloží DNS jméno serveru na binární IP adresu a uloží do proměnné *hostip,* jenž je následně použita ve struktuře s*erver_addr.* Do této struktury musí být uloženo také číslo portu *server_addr.sin_port* a typ protokolu *server_addr.sin_family*.

Poté je nutné svázat soket se jménem, k tomu se používá funkce *bind():*

```
int error = bind(socket_listen, (struct sockaddr*) & server_addr,
        sizeof (server_addr));
```

Funkce *bind()* obsahuje tři parametry, první parametr je file-deskriptor mého vytvořeného soketu, druhý parametr je struktura sockaddr_in a poslední parametr je délka této struktury. Funkce vrací -1 v případě chyby a nulu v případě úspěšného svázání soketu se jménem.

Po úspěšném volaní *bind()* se volá funkce *listen(),* která slouží k nastavení soketu do stavu čekání na příchozí spojení. Má dva parametry, první je opět file-deskriptor vytvořeného soketu a druhý parametr je maximální počet požadavků, čekajících ve frontě. Funkce vrací -1 v případě chyby a nulu v případě úspěchu.

```
int error = listen(socket_listen, 9);
```

Další funkcí potřebnou pro úspěšné vytvoření soketu na straně serveru je funkce *accept(). Tato* funkce čeká na příchozí spojení. Funkce vrátí parametry prvního spojení ve frontě a nový socket, který slouží ke komunikaci s druhou stra*nou.* Má opět tři parametry a jsou shodné jako u funkce bind().

```
int sock_client = accept(socket_listen, (sockaddr*) & ain, &server_size);
```

Poslední neméně důležitou funkcí je funkce close(), která slouží k uzavření soketu, nebo-li k uzavření spojení. Tato funkce má jeden parametr a to file-deskriptor soketu, který chceme uzavřít. Další možností uzavření soketu je funkce shutdown(), obsahuje dva parametry, první je file-deskriptor a druhý parametr je způsob uzavření spojení, a to buď 0 pro uzavření pro příjem, 1 pro uzavření pro vysílaní a 2 pro reset spojení.

6.2 Systémová funkce select()

Jednou z vlastností soketu je tzv. blokovací mód. Není-li ve frontě na spojení žádný požadavek, program se na funkci *accept()* zablokuje do té doby, dokud se neobjeví požadavek na spojení. Tato vlastnost není pro program serveru vhodná, protože program by neměl být neustále v blokovaném stavu, ale měl by být v činnosti. K odstranění této vlastnosti je možné použít tzv. neblokovací režim nebo více vláken.

Já jsem použil v blokovacím režimu metodu seskupení soketů. Je to metoda, kdy se sokety seskupí do tzv. množiny soketů. V Linuxu se pro práci s množinami využívá knihovna *sys/types.h* a v ní definované makra.

Pro vytvoření množiny jsem použil následující: .

```
fd_set read_in;
FD_ZERO(&read_in);
FD_SET(STDIN_FILENO, &read_in);
FD_SET(socket_listen, &read_in);
```

Příkazem *fd_set* se vytvoří datový typ „množina“. Makro *FD_ZERO()* slouží k vyprázdnění množiny, a jako parametr je použit ukazatel na datový typ „množina“. *FD_SET()* je určen k vložení prvku do množiny. První parametr je vkládaný prvek, druhý parametr je ukazatel na datový typ „množina“. V mém programu mám *FD_SET()* jako vkládaný prvek nastaven v jednom případě soket „socket_listen“ a v druhém *STDIN_FILENO*, který kontroluje výstup z klávesnice.

Když je vytvořena „množina“ volá se funkce *select(),* která je určena pro práci s více sokety najednou. Umí pracovat až se třemi skupinami soketů. V první skupině hlídá, zda v některém soketu nejsou příchozí data, která je možno číst. Ve druhé skupině hlídá, zda do některého soketu je možné data zapisovat. Sokety ve třetí skupině jsou kontrolovány, zda se nenacházejí v chybovém stavu. Funkce *select()* vypadá takto:

```
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
        struct timeval *timeout);
```

Parametr *nfds* je největší file descriptory ze všech třech zvětšený o 1. Parametr *readfds* je ukazatel na množinu file descriptoru, z nichž chceme číst. Parametr *writefds* je ukazatel na množinu file descriptorů, do kterých chceme zapisovat. Parametr exceptfds je ukazatel na množinu file descriptorů, u kterých chceme zjistit, zda nastala nějaká chyba. Posledním parametrem je ukazatel na strukturu, udávající, jak dlouho má funkce *select()* tento dohled provádět. Funkce vrací –1 v případě chyby. Jinak vrací počet soketů, na nichž došlo k událostem, které mají být sledovány. V případě, že vyprší časový limit daný posledním parametrem, funkce *select* vrací nulu, protože počet soketů na kterých došlo ke změně, je roven nule.

```
if (select(socket_listen + 1, &read_in, NULL, NULL,&time) == ERROR)
break;
    if (FD_ISSET(STDIN_FILENO, &read_in)) {
      .
      .
    } else if (FD_ISSET(socket_listen, &read_in)) {
      accept()
      .
      .
    }
```

V mém programu funkce select() zablokuje chod programu a čeká, dokud nenastane nějaká událost na soketech pro čtení nebo na výstupu klávesnice. Nebo pokud nevyprší čas určený strukturou *time*. Po skončení volání funkce select() je zjištěno z návratové hodnoty, zda došlo k chybě, nebo na kolika soketech se událo něco rozhodujícího. Jestliže nedošlo k chybě zjistíme funkcí *FD_ISSET(),* zda ke změně došlo na soketu, nebo na výstupu z klávesnice. Pokud je změna zjištěna na soketu, je volána funkce *accept()* a jsou čtena data od klienta. Jestliže je zjištěna změna na výstupu klávesnice, je kontrolováno, co bylo zadáno.

6.3 Systémová funkce fork()

Jakmile jsou přijata data od klienta, je potřeba, aby program dále poslouchal, zda se nevytvoří další spojení a zároveň zpracovával už data přijaté. K tomu je nutné, aby aplikace umožňovala realizovat více činností najednou. Realizace více činností najednou je možné dosáhnou použitím vláken nebo vytvořením podřízených procesů funkcí *fork().*

V programu je po každém přijmutí dat od klienta vytvořen podřízený proces využitím funkce *fork().* Každý takto vytvořený proces zpracuje data, připojí se k LDAP serveru, zkontroluje oprávnění a výsledek pošle klientu. Potom se podřízený proces ukončí.

Pomocí funkce fork() se vytvoří podřízený proces. Nadřízený proces pokračuje v realizaci svého kódu následujícího po volání funkce fork(). Podřízený proces realizuje stejný program od stejného místa. Funkce fork() vrací jiný návratový kód nadřízenému procesu a jiný podřízenému procesu. Návratovou hodnotou v nadřízeném procesu je identifikační číslo podřízeného procesu, návratovou hodnotou v podřízeném procesu je nula. Díky tomuto faktu je možné odlišit nadřízený proces od podřízeného při psaní kódu.

To, že je možné mít naráz spuštěno více procesů, je umožněno neustálým přepínáním procesů v procesoru. Nikdo však nezaručuje, jak dlouho bude procesor zpracovávat proces. Pokud budete mít program realizovaný dvěma procesy, nebude zaručeno, jestli bude dříve zpracován rodičovský, nebo podřízený proces. Může se také stát, že rodičovský proces bude ukončen dříve než proces podřízený. V programu ukončuji každý podřízený program funkcí:

```
int kill(pid_t pid, int sig);
```

Má dva argumenty, první je identifikátor podřízeného procesu a druhý je signál, který

je poslán procesu. V programu je použit signál *SIGKILL*, který značí okamžité ukončení procesu.

I když je proces ukončen, nemusí být automaticky „uklizen“. Rodičovský proces se stará o „uklízení“ podřízených procesů automaticky voláním funkce *wait(),* která ho zablokuje dokud se neukončí podřízený proces. Někdy se ovšem stane, že rodičovský proces nezavolá funkci wait() a z podřízeného procesu se po ukončení stane proces, který je ukončen, ale není „odklizen“ avšak zabírá systémové prostředky. Tomuto je v programu zabráněno následujícími dvěma řádky kódu:

```
int status;
while (!wait(&status));
```

Funkce *wait()* zablokuje rodičovský proces do té doby dokud nedostane návratový kód podřízeného procesu. Potom je podřízený proces ukončen a „odklizen“ a rodičovský proces pokračuje v činnosti.

6.4 Získaní dat z XPortu

Poté co jsem vyřešil komunikaci XPortu se serverem pomocí soketů, mohl jsem přejít na řešení problému získání dat z XPortu. V kapitole 3.2 je popsáno v jakém tvaru pošle XPort data serveru. Problém ovšem nastává u RFID čtečky, která předává XPortu data ne ve standardní ASCII normě. Proto je tedy nutné na straně serveru převést data na standardní ASCII formát. Pokud tedy data z XPortu příjme server, zavolá metodu *vyparsuj_DATA():*

```
char vyparsuj_DATA(char * buf) {
   if (buf[0] != 0x02) {
      log->setLog("Server", "Řetězec nezačíná 0x02");
      return 1;
   }
   if (buf[15] != 0x03) {
      log->setLog("Server", "Řetězec nekončí 0x03");
      return 1;
   }
   for (int i = 0; i < 12; i++) {
      buf[i] = buf[i + 1];
   }
   buf[12] = 0x00;
   return 0;
}
```

Tato metoda dostane jako parametr 16 bajtové pole dat z XPortu. V první podmínce zjistí, jestli začátek pole začíná znakem STX a v druhé podmínce ověří, jestli konec pole končí znakem ETX. Jestliže jedna z těchto dvou podmínek není splněna, znamená to, že data nebyla poslána z RFID čtečky, nebo došlo k chybě během přenosu. Server data zahodí, zapíše informaci o této události do logovacího souboru a čeká na další spojení. Pokud je vše v pořádku pokračuje metoda cyklem, který odstraní z pole první a poslední 3 znaky. Zůstanou pouze data a checksum a na konec pole je přidán znak 0x00, který značí konec pole dat.

Dále server volá metodu *preved_DATA(),* která má za úkol převést data do standardní ASCII normy. Nejprve je otestována podmínka, zdali řetězec končí znakem 0x00. Pokud ano, pokračuje server cyklem, který každý znak v řetězci převede do ASCII normy. Poté opět ukončí řetězec znakem 0x00.

```
char preved_DATA(char * buf) {
   if (buf[12] != 0x00) {
      log->setLog("Server", "String nekončí 0x00");
      return 1;
   }
   for (int i = 0; i < 12; i++) {
      buf[i] = prevod(buf[i]);
   }
   buf[12] = 0x00;
   return 0;
}
```

6.5 Kontrolní součet (Checksum)

Po převedení dat do ASCII normy je dále potřeba zkontrolovat, zda během přenosu nedošlo k poškození dat. K tomu slouží kontrolní součet, což je doplňková informace, která se předává spolu s vlastní informací a slouží ke kontrole, zda vlastní informace je úplná, tedy jestli nedošlo během přenosu k chybě. Použitý obvod AXA012 používá k výpočtu kontrolního součtu exclusive-or (XOR) na pět znaků dat. Jelikož obvod AXA012 přenáší 10 bajtové pole dat, jsou na konec přidané dva bajty kontrolního součtu, pro pět bajtů jeden. V programu server zavolá po přijetí dat a převedení dat metodu *checksum(),* která má dva parametry, pole dat s kontrolním součtem a délku řetězce:

```
char checksum(char *base, int len) {
   unsigned char *buf = (unsigned char *) base;
   unsigned int checksum = 0;
   int i,kod;
   for (i = 0; i < len;i++){
      if((*buf >= '0') &&(*buf <= '9') ){
         kod= *buf-'0';
      }else if((*buf >= 'A') &&(*buf <= 'F')){
         kod=10 + *buf-'A';
      }
        checksum ^= kod;
        *(buf++);
        kod=0;
   }
   if (checksum != 0) {
     log->setLog("Server", "Neplatný checksum");
      return 1;
   } else { base[10] = 0x00; }
   return 0; }
```

Celá metoda pracuje velmi jednoduše, základem je cyklus, který prochází řetězec znak po znaku a každý znak převádí do dekadické formy tak, aby znak odpovídal hexadecimální formě. Takže znaky A – F odpovídají číslicím 10 – 15. Následně je na každý převedený znak aplikován XOR a výsledek je uložen do proměnné *checksum.*

Po ukončení cyklu, kdy na celý řetězec i s dvěma bajty kontrolního součtu je aplikován logický XOR, musí být proměnná *checksum* rovna nule. Pokud je výsledek roven 0 znamená to, že během přenosu nedošlo k chybě, tudíž dojde k odstranění dvou bajtů kontrolního součtu a data jsou zahešována a poslána na LDAP server.

Ovšem pokud má výsledek jinou hodnotu než nulu, tak to značí, že došlo během přenosu k chybě, z tohoto důsledku jsou data zahozena, server zapíše informace o události do souboru a dostane se do stavu čekání na další spojení od klienta.

6.6 Přístup k LDAP serveru

Princip LDAP serveru je stručně popsán v kapitole 2. V této části budu popisovat možnosti přístupu k LDAP serveru. Jak už bylo zmíněno, přístup k LDAP serveru je potřeba pro získání informací o osobě. Tato osoba přiloží svoji identifikační kartu k RFID čtečce, kde je následně sejmut identifikační řetězec a ten je po zavolání hašovací funkce poslán na LDAP server, kde jsou získané informace o dané osobě.

První a poměrně jednoduchý případ jak se připojit k LDAP serveru je utilita *ldapsearch*. Tato utilita se nainstaluje jako součást balíku *ldap-utils* balíčkového systému distribuce Debian. Dále je nutné v adresáři */etc/ldap/* změnit konfigurační soubor *ldap.conf* tak, že je přidán řádek:

```
TLS_REQCERT    never
```

Poté stačí zadat:

```
ldapsearch -h ldap.vsb.cz -x cn=os-cislo
```

a zobrazí se veškeré informace o dané osobě. Tato možnost je velice jednoduchá a z programátorského hlediska nenáročná na výsledný kód. Stačí, pokud utilitu *ldapsearch* spojíme v programovacím jazyce s funkcí *execlp(),* asi takto:

```
execlp("ldapsearch","ldapsearch", "-h","ldap.vsb.cz","-x",
"cn=os-cislo",NULL);
```

Funkce *execlp()* nespouští příkazový interpret, ale jen daný program. První dva argumenty jsou cesty k programu určené systémovou proměnou PATH. Zbylé argumenty funkce *execlp()* jsou argumenty pro spuštění volaného programu. Poslední argument funkce *execlp()* musí být NULL.

I když je to řešení poměrně snadné, je tento způsob nevhodný pro můj program. A to z důvodu nemožnosti nastavení vlastností připojení a vyhledávání v LDAP serveru. Například nemůžeme nastavit timeout spojení v případě nefunkčnosti LDAP serveru, dále volání *ldapsearch* přes funkci *execlp()* je pomalé. To jsou důvody, proč tento způsob komunikace z LDAP serverem v mém programu nevyužívám.

Další možnosti přípojení k LDAP serveru je použití knihovny *ldap.h* přímo v programovacím jazyce C++. Tato knihovna obsahuje funkce pro připojení k LDAP

serveru a následném vyhledávání, přidávání, editování a mazání položek v adresářové struktuře. Tedy je možné mít od počátku spojení až po ukončení spojení plnou kontrolu nad prováděnými příkazy.

Pro funkčnost knihovny *ldap.h* je potřeba nejprve nainstalovat balíčky *libldap* a l*ibldap-dev*. Tyto balíčky jsou nutné připojení k LDAP serveru a následný vývoj v programovacím jazyce C++.

Ke komunikaci s LDAP serverem slouží třída *AutorizaceLDAP.cpp*, které je předán hašovaný řetězec. Tento řetězec je předán jako argument metodě *pripojLdap()*, která vytváří spojení s LDAP serverem. K vytvoření spojení je použita funkce:

```
int ldap_initialize(LDAP *ld, char *uri);
```

Tato funkce má dva argumenty - LDAP strukturu, která je voláním této funkce alokována a URI (Uniform Resource Identifier), nebo-li jednotný identifikátor zdroje, což je řetězec znaku definující LDAP server.

Dále je možné nastavit různé vlastnosti LDAP spojení, a to pomocí funkce:

```
int ldap_set_option(LDAP  *ld, int  optionToSet, const void *optionValue );
```

Funkce má tři argumenty, první je LDAP struktura, která byla alokována ve funkci *ldap_initialize()*. Druhý argument je seznam možností, které můžeme nastavit pro LDAP spojení. V mém programu jsem použil možnost *LDAP_OPT_NETWORK_TIMEOUT*, která indikuje maximální počet sekund, po které čeká program na odpověď od LDAP serveru. Těchto možností je mnohem více, avšak všechny je jmenovat nebudu. Poslední argument je adresa hodnoty, která se týká vybrané vlastnosti spojení. V mém případě je poslední argument roven pěti a znamená to, že program čeká maximálně pět sekund na odpověď od LDAP serveru a poté ruší celé spojení.

Po úspěšném navázání spojení a nastavení vlastností spojení se volá funkce:

```
int ldap_sasl_bind_s(
        LDAP            *ld,
        const char      *dn,
        const char      *mechanism,
        const struct berval  *cred,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        struct berval   **servercredp).
```

Úkolem této funkce je ověřování klienta na serveru. Pracuje na principu SASL (Simple Authentication and Security Layer) což je obecná metoda pro přidávání, nebo zlepšování ověřování v protokolech klient/server. Pokud se nastavuje SASL, musí se rozhodnout pro ověřovací mechanismus, pro výměnu ověřovacích informací a *ověřovací* systém pro uložení informací o uživatelích.

První argument je stejný jako u předešlých funkcí LDAP struktura. Druhý argument obsahuje identifikaci objektu, v mém případě je to

"cn=tuo_card_id_reader,ou=home,o=cvt" avšak může mít i hodnotu NULL. V třetím argumentu se volí mechanismus ověřování, já jsem zvolil hodnotu NULL, tedy nedochází k žádnému ověřování. Čtvrtý argument obsahuje strukturu, která obsahuje heslo, pokud je heslo použito.

Pátý argument specifikuje seznam možných řídících prvků pro LDAP server. Šestý argument je ten shodný s pátým, akorát pro LDAP klienta. Tyto dva argumenty mohou být nastaveny na hodnotu NULL. Poslední argument nastavuje parametr ověřování, vráceným LDAP serverem. Pokud server nevrací žádné ověření, bude nastaveno na hodnotu NULL. Jelikož nepoužívám žádné ověřování, mám poslední tři argumenty ve funkci ldap_sasl_bind_s() nastaveny na NULL hodnotu.

Pokud funkce bind proběhla v pořádku, je volána další funkce pro synchronní vyhledávání:

```
int ldap_search_ext_s(
    LDAP          *ld,
    const char    *base,
    int           scope,
    const char    *filter,
    char          **attrs,
    int           attrsonly,
    LDAPControl   **serverctrls,
    LDAPControl   **clientctrls,
    struct timeval  *timeout,
    int           sizelimit,
    LDAPMessage   **res).
```

Tato funkce má dohromady jedenáct argumentů. První je LDAP struktura určena funkcí *ldap_initialize(). A*rgument *base* představuje DN záznam, na kterém se začne vyhledávat. V programu je použita hodnota NULL. Argument *scope* specifikuje způsob vyhledávání. K dispozici jsou tři možnosti. Možnost *LDAP_SCOPE_BASE,* která vyhledává samotný objekt, pak možnost *LDAP_SCOPE_ONELEVEL* vyhledávající přímé potomky hledaného objektu a poslední možnost *LDAP_SCOPE_SUBTREE,* která vyhledá objekt a všechny její následovníky.

Argument *filter* specifikuje filtr nebo také řetězec, který se bude hledat na LDAP serveru, V programu je to haš řetězec z RFID čtečky ve tvaru *TUOCardMD5=<haš řetězec>*. Argument *attrs* specifikuje, které typy atributů budou vráceny z LDAP serveru. Pokud je zvolena hodnota NULL, znamená to, že budou vráceny všechny typy atributu výsledku hledaní. U argumentu *attrsonly* se nastavuje hodnota 1 pro vrácení pouze typů atributů nebo hodnota 0 pro vrácení jak typů atributů tak hodnot jednotlivých atributů záznamu. Argument *serverctrls* a *clientctrls* nastavuje seznam řídicích prvků pro LDAP server a klienta. Argument *sizelimit* udává maximální počet vrácených záznamů. Argument *timeout* nastavuje časový limit, který je posílán LDAP serveru. Poslední argument *res* obsahuje výsledek synchronní vyhledávající operace.

Všechny uvedené funkce pro LDAP vrací v případě úspěšného provedení hodnotu *LDAP_SUCCESS.* V případě neúspěchu vrací funkce chybu, kterou je možné interpretovat funkcemi l*dap_perror()* nebo *ldap_err2string().*

K procházení výsledku z funkce *ldap_search_ext_s(),* jsou určeny funkce *ldap_first_entry()* a l*dap_next_entry().* První z nich vrací ukazatel na první záznam výsledku a druhá vrací ukazatel na následující záznam po funkci *ldap_first_entry()* nebo l*dap_next_entry().* Obě funkce mají dva argumenty, LDAP strukturu a ukazatel na výsledek *res* z funkce *ldap_search_ext_s()*.

Další funkce nezbytné pro procházení jednotlivými záznamy jsou :

```
char *ldap_first_attribute(
        LDAP        *ld,
        LDAPMessage  *entry,
        BerElement  **berptr).
```

Tato funkce vrací ukazatel na první atribut záznamu. Obsahuje tři argumenty, první je stejný jako u všech funkcí knihovny *ldap.h.* Argument *entry* je výsledek vracený funkcemi *ldap_first_entry()* nebo l*dap_next_entry().* A poslední argument *berptr* obsahuje ukazatel na strukturu BerElement. Dále funkce *ldap_next_attribute()* je ekvivalentní funkci *ldap_first_attribute()* až na to, že vrací další atribut v záznamu.

K získání hodnot jednotlivých atributů je zapotřebí volat funkci:

```
struct berval **ldap_get_values_len(
        LDAP        *ld,
        LDAPMessage  *entry,
        const char   *attr).
```

Obsahuje opět tři argumenty. Argument *entry* je výsledek vrácený funkcemi *ldap_first_entry()* nebo l*dap_next_entry()* a argument *attr* specifikuje název typu atributu u něhož je potřeba znát hodnotu. Výsledek je uložen do struktury berval:

```
struct berval {
    unsigned long bv_len;
    char  *bv_val;
};
```

Struktura berval obsahuje *bv_len,* kde je uložena délka vrácené hodnoty a *bv_val*, která obsahuje ukazatel na hodnotu.

Po získaní všech hodnot je nutné uvolnit zdroje. Pro uvolnění paměti alokovanou funkcí *ldap_get_values_len()* se používá funkce:

```
void ldap_value_free_len( struct berval   **vals).
```

K uvolnění paměti struktury BerElementu alokovanou funkcemi *ldap_next_attribute()* a *ldap_first_attribute()* je použita funkce:

```
void ber_free(
  BerElement *pBerElement,
  INT fbuf).
```

Argument *pBerElement* obsahuje ukazatel na BerElement strukturu, kterou chceme uvolnit a argument *fbuf* obsahuje hodnotu 0 jestliže je potřeba uvolnit BerElement strukturu alokovanou funkcemi *ldap_next_attribute()* a *ldap_first_attribute(),* jinak obsahuje hodnotu 1.

Paměť alokovanou funkcí *ldap_search_ext_s()* se uvolní funkcí, která má jako argument ukazatel na LDAP message alokovanou *ldap_search_ext_s():*

```
int ldap_msgfree(LDAPMessage  *msg).
```

Pro ukončení synchronního spojení s LDAP serverem se používá funkce:

```
int ldap_unbind_ext_s(LDAP *ld,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls).
```

Tato funkce umožňuje uzavřít spojení jak na straně LDAP serveru, tak na straně klienta. Obsahuje tři argumenty. Argument *ld* obsahuje ukazatel na LDAP strukturu vrácenou funkcí *ldap_initialize().* Argumenty *serverctrls* a *clientctrls* nastavují seznamy řídících prvků pro LDAP server a klienta.

6.7 Konfigurační soubor

Pro dobrou konfigurovatelnost výsledného programu je vhodné použít konfigurační soubor, který bude obsahovat základní informace potřebné pro běh programu. Výhodou konfiguračního souboru je možnost měnit data potřebná ke spuštění programu, aniž by bylo nutné kompilovat celý program. Stačí pouze změnit údaje v konfiguračním souboru a program po spuštění načte údaje ze souboru.

Nejdříve jsem si musel promyslet jaký konfigurační soubor budu v mém programu používat. Nabízela se možnost použít obyčejný textový soubor a v něm mít uložena data. Pak k němu přistupovat pomocí souborových streamu. Tento způsob konfigurace jsem posléze opustil pro jeho složitost, sekvenční procházení souboru a nemožnosti efektivního vytváření sekcí v klasickém souboru. Další možnost bylo spouštět program s parametry, podle kterých by se spustil skript pro načtení hodnot ze souboru.

Nakonec jsem se rozhodl pro parsrovací knihovnu libConfuse. Tato knihovna obsahuje mnohé funkce pro práci s konfiguračními soubory. S touto knihovnou je možné vytvářet seznamy pro různé typy hodnot (string, integer, float, boolean) nebo také sekce, které umožňují dělit nastavení do logických bloků. Dále je jednoduché, podle mého názoru, vytvoření konfiguračního souboru a následné upravování konfiguračního souboru.

K tomu, aby bylo možné knihovnu libConfuse používat, je nutné nainstaloval knihovny *libconfuse0* a *libconfuse-dev.* Dále v programu je nutné importovat knihovnu *confuse.h*. V mém programu se o konfigurační soubor stará třída *Conf.cpp.*

Zde je jednoduchá ukázka mého konfiguračního souboru *configure.conf*:

```
port=10001
ip_server="192.168.2.102"
smtp_server="smtp.vsb.cz"
smtp_port=25

class A1033{
ip="A1033"
allow={"cn=STU_FEI,ou=FEI,ou=GROUPS,o=VSB"}
deny={"mol049"}
}
```

Tento soubor obsahuje položky nutné pro běh programu. Obsahuje položku *port,* na kterém bude server poslouchat. Dále položku *ip_server,* ze které server načte svojí IP adresu. Položky smtp_port, smtp_server pro připojení ke školnímu poštovnímu serveru. Dále obsahuje sekce pro každou místnost, kde je používána čtečka. Tyto sekce jsou označeny *class <číslo místnosti>*. Každá sekce obsahuje IP adresu XPortu, ke kterému je přidělena daná sekce. Položku *allow* což je seznam osob, které mají oprávnění vstoupit do místnosti a položku *deny,* která má přesně opačný účel jako položka *allow.* Položky *allow a deny* obsahují seznamy hodnot LDAP atributů.

Pro inicializaci konfiguračního souboru ve třídě *Conf.cpp* je vytvořena metoda *parse_Conf().* Nejdříve bylo nutné inicializovat jednotlivé sekce v konfiguračním souboru:

```
cfg_opt_t class_opts[]={

CFG_STR_LIST("allow","{NULL}",CFGF_NONE),
    CFG_STR_LIST("deny","{NULL}",CFGF_NONE),
    CFG_STR("ip","NULL",CFGF_NONE),
    CFG_END()
  };
```

K vytvoření sekcí je použita datová struktura *cfg_opt_t,* která obsahuje makra k inicializaci seznamů *allow* a *deny* a položky *ip.* Každé makro má tři parametry, název položky, implicitní hodnotu a hodnotu flagu. Flag *CFG_END()* ukončuje seznam hodnot, musí být na konci struktury *cfg_opt_t.*

Po inicializaci sekcí je nutné inicializovat konfigurační soubor jako celek, to se provede následovně:

```
cfg_opt_t opt[]={
   CFG_INT("port",0,CFGF_NONE),
   CFG_INT("smtp_port",0,CFGF_NONE),
   CFG_STR("ip_server","NULL",CFGF_NONE),
   CFG_STR("smtp_server","NULL",CFGF_NONE),
   CFG_SEC("class",class_opts,CFGF_TITLE | CFGF_MULTI),
   CFG_END()
};
```

Opět se použije struktura *cfg_opt_t* obsahující makra pro vytvoření jednotlivých položek. Makro *CFG_SEC()* kromě názvu položky obsahuje také název *cfg_opt_t* struktury, která definuje sekce. Flag *CFGF_TITLE* znamená, že tato položka má titulek a *CFGF_MULTI* určuje, že položka může být mnohonásobně specifikovaná. Tyto dva flagy jsou určeny pouze pro definování sekcí.

Dále je nutné inicializovat kompletní konfigurační soubor, aby ho bylo možné parsrovat:

```
cfg =cfg_init(opt,CFGF_NONE);
if(cfg_parse(cfg,soubor)==CFG_PARSE_ERROR){
   log->setLog("conf","Chyba v konfiguracnim souboru");
   exit(EXIT_FAILURE);
}
```

Nejprve se zavolá funkce *cfg_init(),* která má dva argumenty, název struktury *cfg_opt_t* a hodnotu flagu. Tato funkce vrací *cfg_t* strukturu, která je následně použita ve funkci *cfg_parse().* Také má dva argumenty, první je *cfg_t* struktura, druhý argument je název konfiguračního souboru. V případě úspěchu vrací *CFG_SUCCESS*, v případě chyby vrací CFG_PARSE_ERROR.

Třída *Conf.cpp* obsahuje také metodu *kontrola_IP(char* ip_client),* která slouží k přiřazení učebny z konfiguračního souboru k IP adrese XPortu. Program volá tuto metodu vždy, když se k němu připojí XPort. Metoda vypadá takto:

```
kontrola_IP(char* ip_client){
    int j;
   cfg_t *vysledek=NULL;
   for(j=0;j<cfg_size(cfg,"class");j++)
   {
      cfg_class=cfg_getnsec(cfg,"class",j);
      if(strcmp(ip_client,cfg_getstr(cfg_class,"ip"))==0)
      {
         vysledek =cfg_class;
      }
   }
   return vysledek;
}
```

Metoda má jeden argument, který představuje IP adresu XPortu. Úkolem metody je cyklicky procházet konfigurační soubor a pomocí funkce *cfg_getsec()* načítat jednotlivé sekce, které odpovídají učebnám. Potom v každé sekci kontrolovat jestli IP adresa z XPortu odpovídá IP adrese v dané sekci. K načtení IP adresy z každé sekce slouží funkce *cfg_getstr()*. Pokud je nalezena shoda, je vrácen název dané sekce pro další použití. Jestliže je vrácená hodnota NULL, znamená to neoprávněný přístup.

6.8 Syslog

Aby vytvořený program splňoval podmínky zadání, bylo nutné vymyslet vhodný způsob, jakým bude systém ukládat informace o událostech vyvolanými během

programu. Například budeme požadovat dohledání informací o vstupech do učeben několik dní nazpět. Také při pádu programu můžeme dohledat příčiny nestandardního ukončení.

K tomuto účelu slouží takzvané logy, soubory do kterých se ukládají potřebné informace ve vhodné formě pro další zpracování. Jednou z možných řešení je použít textový soubor a ukládat informace v nějakém vhodném formátu. Tuto možnost jsem nevyužil, místo toho jsem použil knihovnu *syslog.h.*

Syslog pracuje na principu komunikace klient – server. *Syslog* deamon (server) přijme zprávy od programu (klient). Potom syslog server k zprávě přidá datum a čas a uloží do souboru. Výhodou tohoto řešení je, že Linux má standardně nastavený syslog a deamon *syslogd* je automaticky spuštěn při startu operačního systému. Dále můžeme nastavovat v souboru */etc/syslog.conf,* kam se budou zprávy ukládat a od jakých programů budou zprávy protokolovány. Pro tento program je důležitá položka:

```
user.*                  -/var/log/user.log
```

Tato položka definuje název a místo uložení souboru. Název a umístění je zcela libovolné, já jsem ovšem ponechal základní nastavení.

O zápis do souboru přes syslog se stará třída *Log.cpp, která* obsahuje jednu metodu:

```
setLog(char* ident, char* message){
   openlog(ident,LOG_PID|LOG_CONS,LOG_USER);
   syslog(LOG_INFO,message);
   closelog();
}
```

Metoda má dva argumenty *ident,* což je identifikátor určující zdroj zprávy a vlastní tělo zprávy *message.* Nejdříve se volá funkce *void openlog(const char *ident, int logopt, int facility)*, která otevírá spojení syslog deamonu. Má tři argumenty, identifikátor zdroje, dále číselné konstanty. *LOG_PID* určuje, že se bude ukládat ID procesu každé zprávy a *LOG_CONS* značí ukládání systémových chyb. Poslední argument určuje, kde se budou zprávy ukládat.

Dále se volá funkce *void syslog(int priority, const char *message,),* která se stará o zaznamenání zprávy. První argument je priorita zprávy, já mám nastaveno *LOG_INFO,* což znamená obecný druh zprávy. Druhý argument je samotná zpráva určená k uložení. Nakonec je volána funkce pro ukončení spojení *closelog()*.

6.9 Implementace protokolu SMTP

Pokud osoba bude chtít vstoupit do místnosti a nebude mít oprávnění, bude mu zaslána informace o neoprávněném přístupu na jeho e-mail. Informace bude obsahovat čas a místo neoprávněného vstupu a dále kontakt na správce systému u kterého může žádat o oprávnění přístupu k místnosti.

Pro odesílaní e-mailu na školní poštovní server je použit protokol SMTP (Simple Mail Transfer Protocol). Protokol zajišťuje doručení pošty do tzv. poštovní schránky adresáta, ke které potom může uživatel kdykoli (off-line) přistupovat (vybírat zprávy) pomocí protokolů POP3 nebo IMAP. SMTP pracuje nad protokoly TCP a na portu 25.

Z hlediska programátora je důležitá forma komunikace mezi klientem a serverem. Zde je ukázka komunikace programu se školním serverem. C: znamená zprávy odeslané klientem a S: zprávy odeslané serverem.

C: HELO smtp.vsb.cz
S: 250 Hello smtp.vsb.cz
C: MAIL FROM: <nobody@vsb.cz>
S: 250 Ok
C: RCPT TO: <osoba@vsb.cz>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: Subject: TUOCard
C:
C: Text zprávy
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye

V programu se o posílaní e-mailů stará třída *Mail.cpp.* Tato třída má za úkol vytvořit pomocí soketu spojení s poštovním serverem. Adresa poštovního serveru je načtena z konfiguračního souboru. Po úspěšně vytvořeném spojení posílá jednotlivé řetězce znaků poštovnímu serveru, jak je znázorněno na ukázce SMTP komunikace mezi klientem a serverem. Po každém odeslání čeká program na potvrzení úspěšného přijmutí zprávy od serveru. Jak odesílatel, tak tělo zprávy je načítáno z konfiguračního souboru.

6.10 Nevýhody vytvořeného systému

Nevýhoda stávajícího návrhu softwaru je, že program serveru nedokáže zjistit zda nedošlo k odpojení modulu s XPortem. Tedy pokud dojde k přerušení a někdo se pokusí o přístup do místnosti, data s RFID čtečky zůstanou na XPortu, který se bude pokoušet po určitou dobu připojit k serveru. Po následné opravě spojení budou data poslána na server a ten bude pokračovat v činnosti i když data nejsou už aktuální a pošle signál XPortu aby otevřel danou místnost. To znamená, že by mohlo dojít ke vstupu do místnosti neoprávněnou osobou.

Tomuto by se dalo předejít tím, že by první data poslána z RFID čtečky byla zahozena a žádána nová data. Ovšem uživatel by musel opakovaně žádat o vstup a to by se mu pravděpodobně nelíbilo. Další možností by bylo periodicky kontrolovat spojení na jednotlivé XPorty, což by bylo zase náročné na rychlost systému.

7 Závěr

Při psaní této práce jsem mohl využít nabyté zkušenosti z předchozího studia a navrhnout systém pro ovládání dveřního zámku za pomoci RFID karet. Tento systém je vytvořen pomocí programovacího jazyka C++ a pracuje pod operačním systémem Linux. Hardware byl dodán panem Ing. Davidem Seidlem.

Systém pracuje jako server, na který se připojuje RFID čtečka. Data z RFID čtečky jsou převedena na Ethernet pomocí XPortu a poslána na server. Server se následně připojí na školní LDAP server a zjistí informace o osobě, která žádá přístup do místnosti. Podle konfiguračního souboru je poté buď povolen nebo zamítnut přístup osobě do místnosti.

Snahou bylo, aby výsledný systém byl jednoduchý, stabilní a v neposlední řadě dobře konfigurovatelný. K tomu jsem použil metody, které jsou popsané v předešlých kapitolách.

Literatura

[1] Gerald Carter:
LDAP System Administration
Vydavatelství O'Reilly Media, 2003
ISBN 1-56592-491-6

[2] Dr. Klaus Finkenzeller:
RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification, 2nd Edition
Vydavatelství Wiley, 2003
ISBN 978-0-470-84402-1

[3] Jiří Sitera:
Adresářové služby - úvod do problematiky
http://www.cesnet.cz/doc/techzpravy/2000-4/

[4] **Dokumentace k OpenLDAP**
http://www.openldap.org

[5] **Dokumentace ke knihovně libConfuse**
http://www.nongnu.org/confuse/

[6] **XPort**
http://www.lantronix.com

[7] Jakub Matys:
Programování pod Linuxem pro všechny
http://www.root.cz/serialy/programovani-pod-linuxem-pro-vsechny/

[8] **Simple Mail Transfer Protocol**
http://www.faqs.org/rfcs/rfc821.html

Obsah přiloženého CD-ROM

doc/	-adresář obsahuje datasheety k XPortu a obvodu AXA012
source/	-adresář obsahuje zdrojové kódy programu
text/	-adresář obsahuje text bakalářské práce ve formátu PDF a OpenOffice
install.txt	-popis spuštění programu